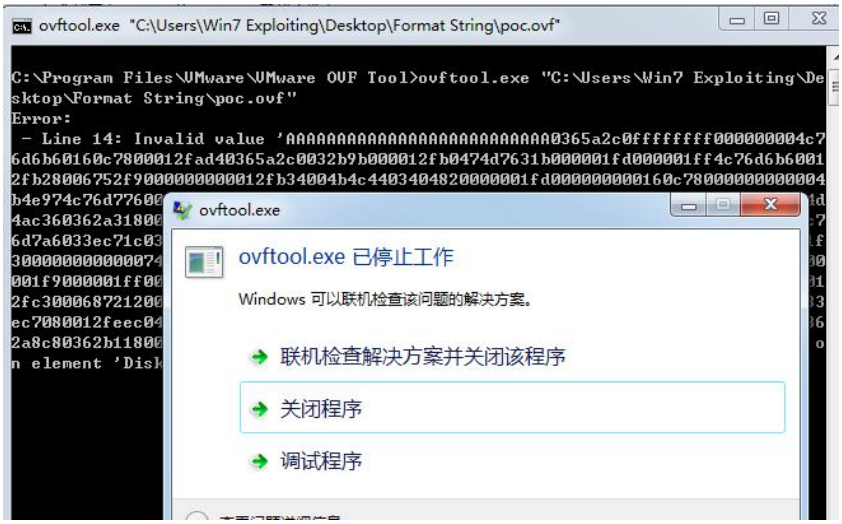
二进制漏洞挖掘系列(9)-实例分析格式化字符串漏洞

WritenBy 東

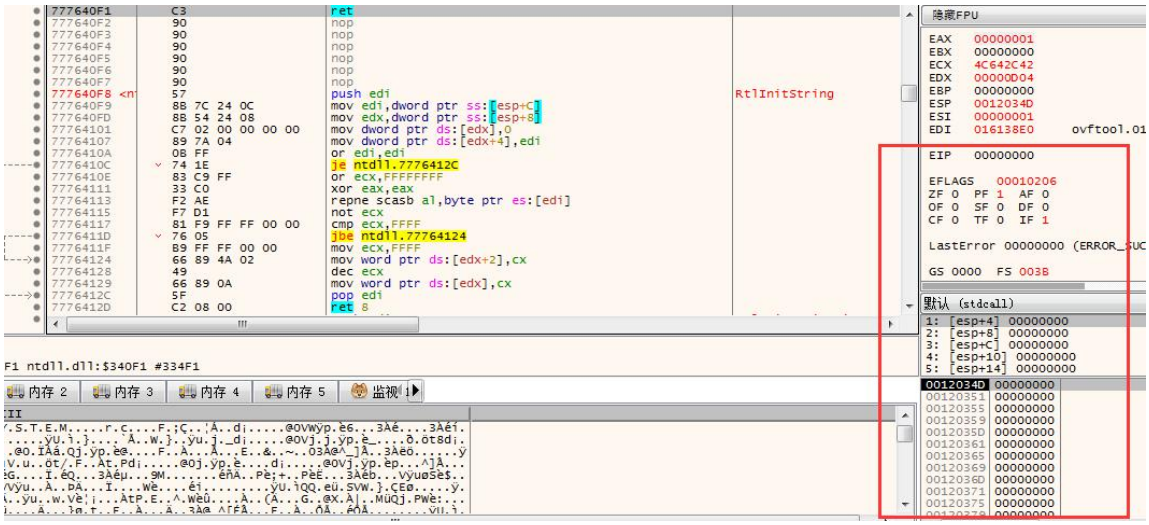
本节例子是 CVE-2012-3569 VMware OVF Tool 格式化字符串漏洞 分析思路来自漏洞战争
实验环境: win7 x86
实验工具: IDA pro, x32dbg,
漏洞软件: VMware-ovftool-2.1.0-467744-win-i386
漏洞战争配套下载资料: http://pan.baidu.com/s/1bo6iwcn

VMware OVF Tool 是由 VMware 免费提供的一款支持命令行运行的导入导出工具,ovftool.exe 在解析 OVF 文件时存在格式化字符串漏洞,攻击者可以诱使用户加载恶意构造的 OVF 文件实现远程任意代码执行。

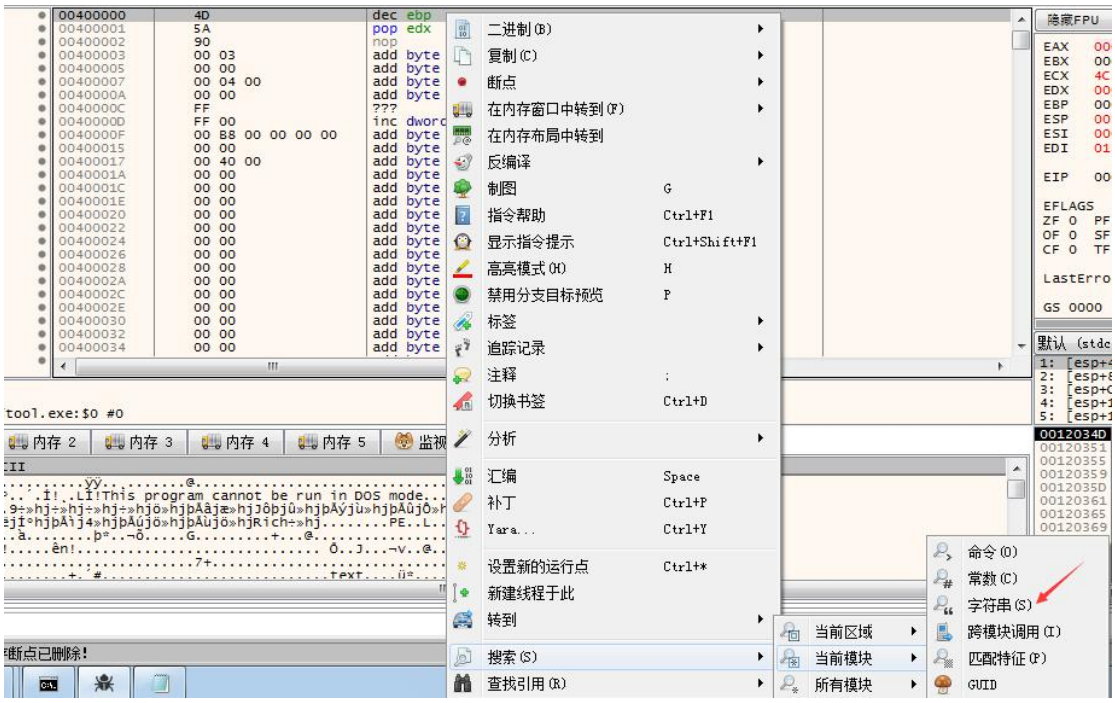
命令行加载程序提供的 poc 程序异常,然后选择调试,我已经将 x32dbg 设置成默认调试器

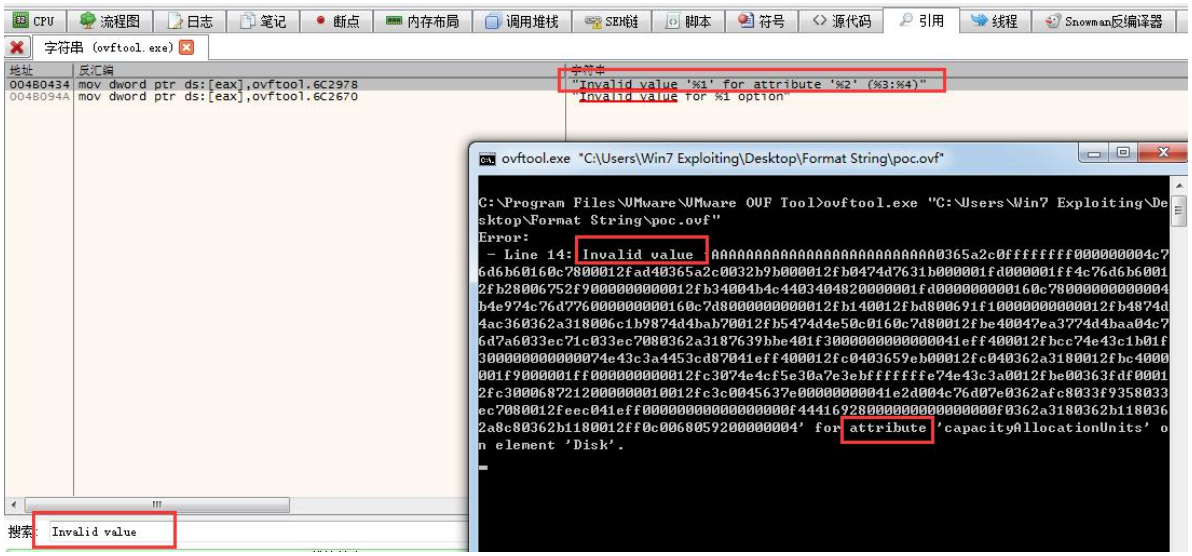


我们发现栈数据全被破坏掉了,也无法通过栈回溯来定位漏洞函数

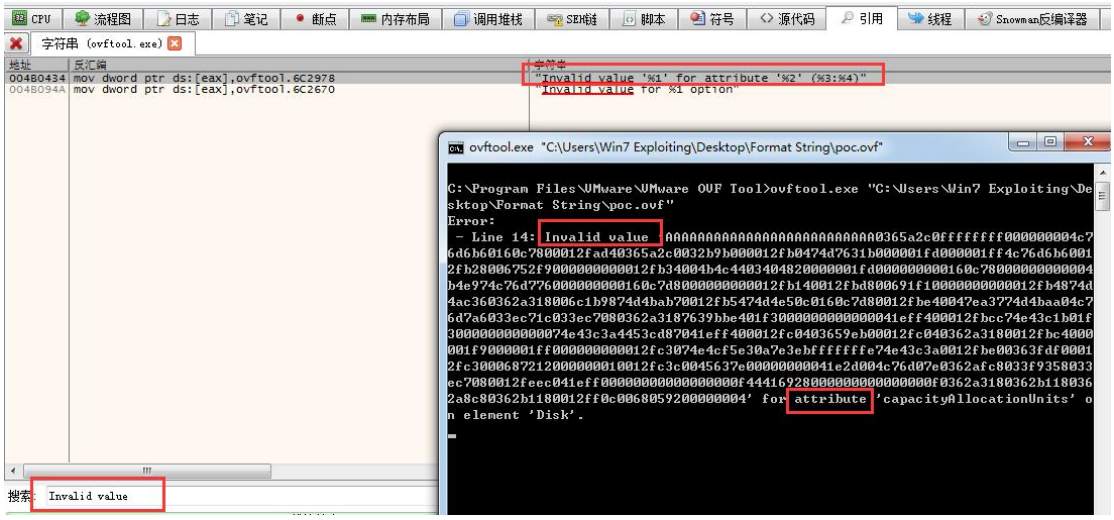


那么则根据程序的输出来定位到这个打印输出的位置,发现输出了 Invalid value 接着我们用字符串搜索功能来搜索 内存布局->反汇编转到 进入 ovftool.exe 领空





在反汇编窗口中转到,这个位置

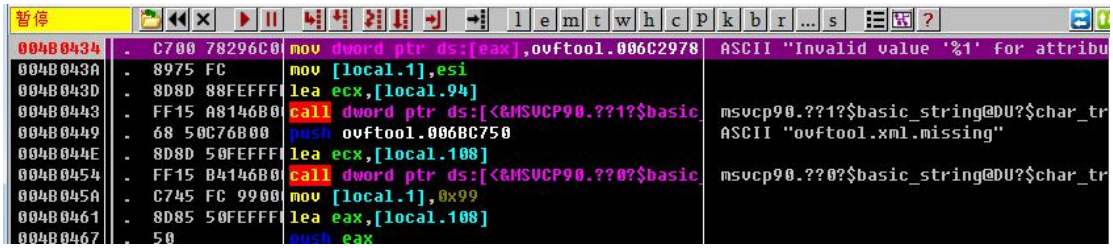


发现这段指令是用于分配字符串

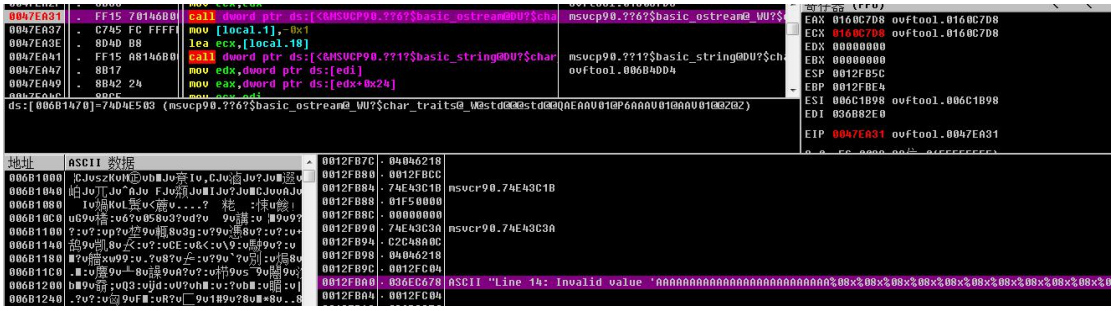
没弄清 x32dbg 怎么加命令行参数 如果你知道请告诉我下 谢谢..

打开 OllyDbg 命令行参数很简单 就是把 poc.ovf 放在 ovetool.exe 的同目录下,启动即可

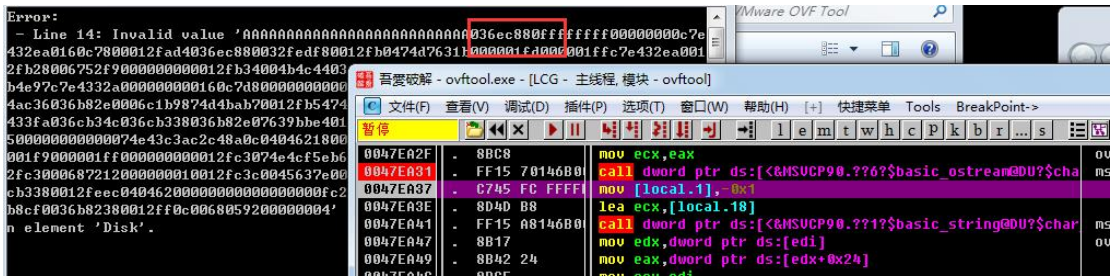
然后 bp 004B0434 F9 运行,断在这里之后单步跟踪调试直到控制台输出了 poc 文件中的 ovf:capacityAllocationUnits 的属性值



单步跟踪到这个地址 0047EA31 观察堆栈 就是我们的打印函数



单步步过这个 call

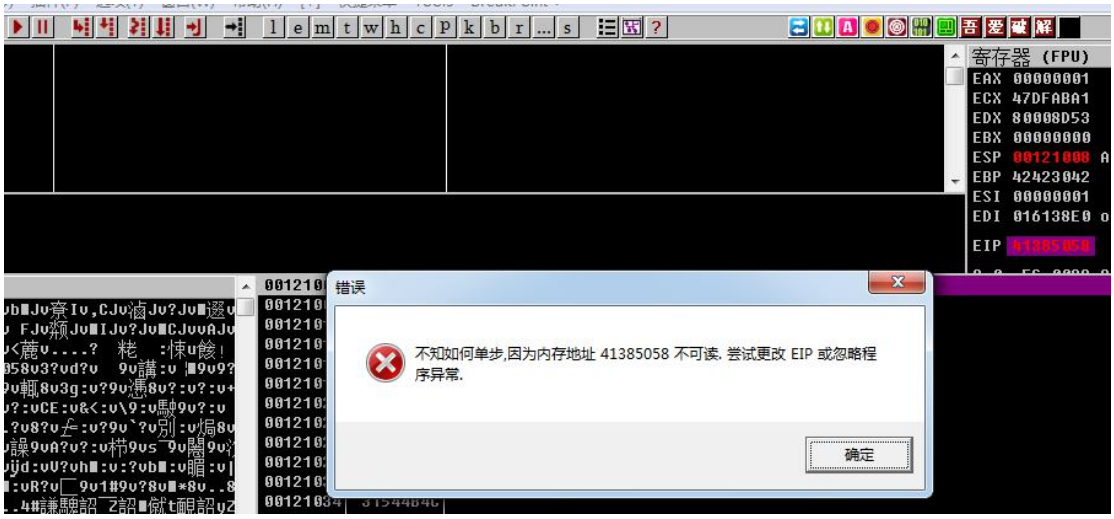


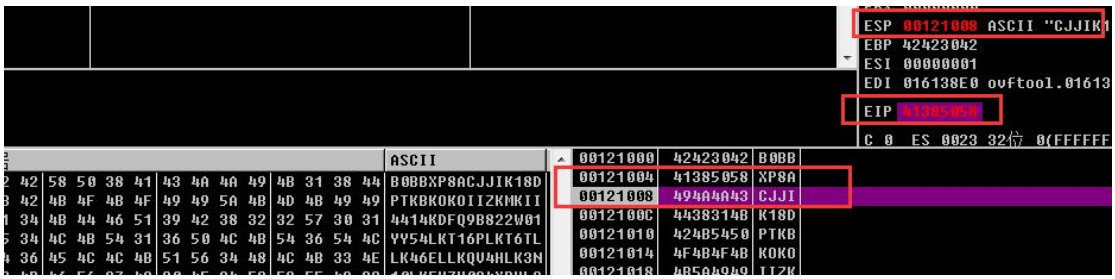
当执行完这个 call 程序的堆栈被破坏,这个函数未对输入的参数进行判断 所以输入格式化字符时会导致遍历栈数据,内存写入 用 IDA 来还原一下这个 call 的伪代码

```
    v23 = v22;
    v24 = sub_401A90(&dword_160C7D8, " - ");
    v25 = std::operator<<<char,std::char_traits<char>,std::allocator<char>>(v24, v23);
    std::basic_ostream<char,std::char_traits<char>>::operator<<(v25, std::endl);
    v31 = -1;
    std::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string<char,std::char_traits<char>,std::allo
    ++v19;
    result = (*(int (__thiscall **)(_DWORD))(*(_DWORD *)v4 + 36))(v4);
  }
  while ( v19 < result );
}
```

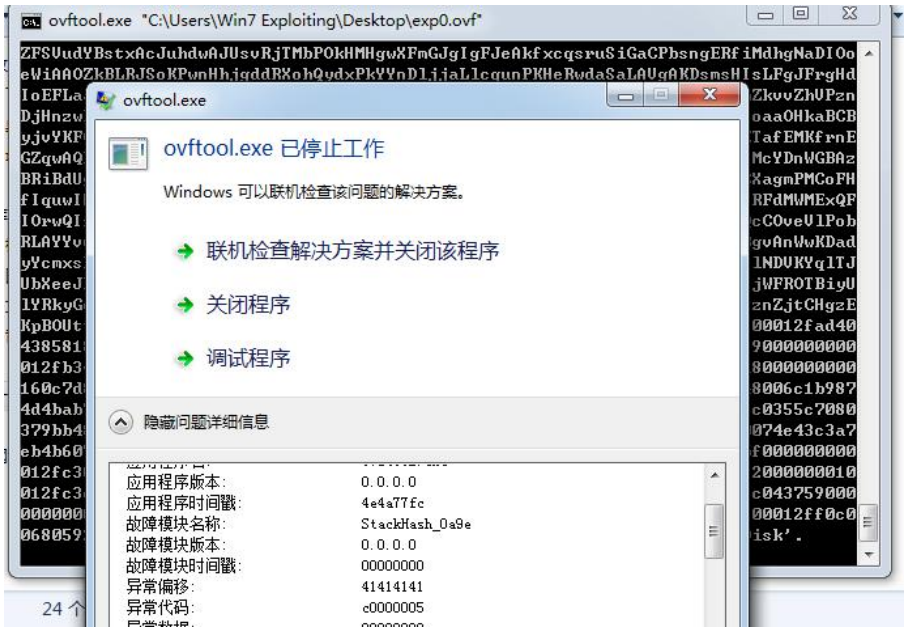
v25 是我们构造的数据,在处理字符串的时候没有加以有效过滤所以产生了漏洞

接下来就是尝试构造 exp,通过上节我们就可以利用%x 控制字符输出数量 %n 来写入返回地址 先用漏洞战争里的 exp 实验一下,内存地址 41385058 很明显是字母的 ASCII 码 XP8A

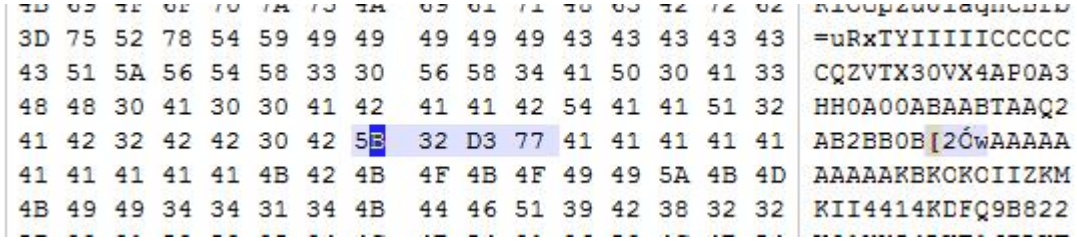




定位到文件中 XP8A 替换成 AAAA 重新运行程序发现崩溃的位置正是 41414141

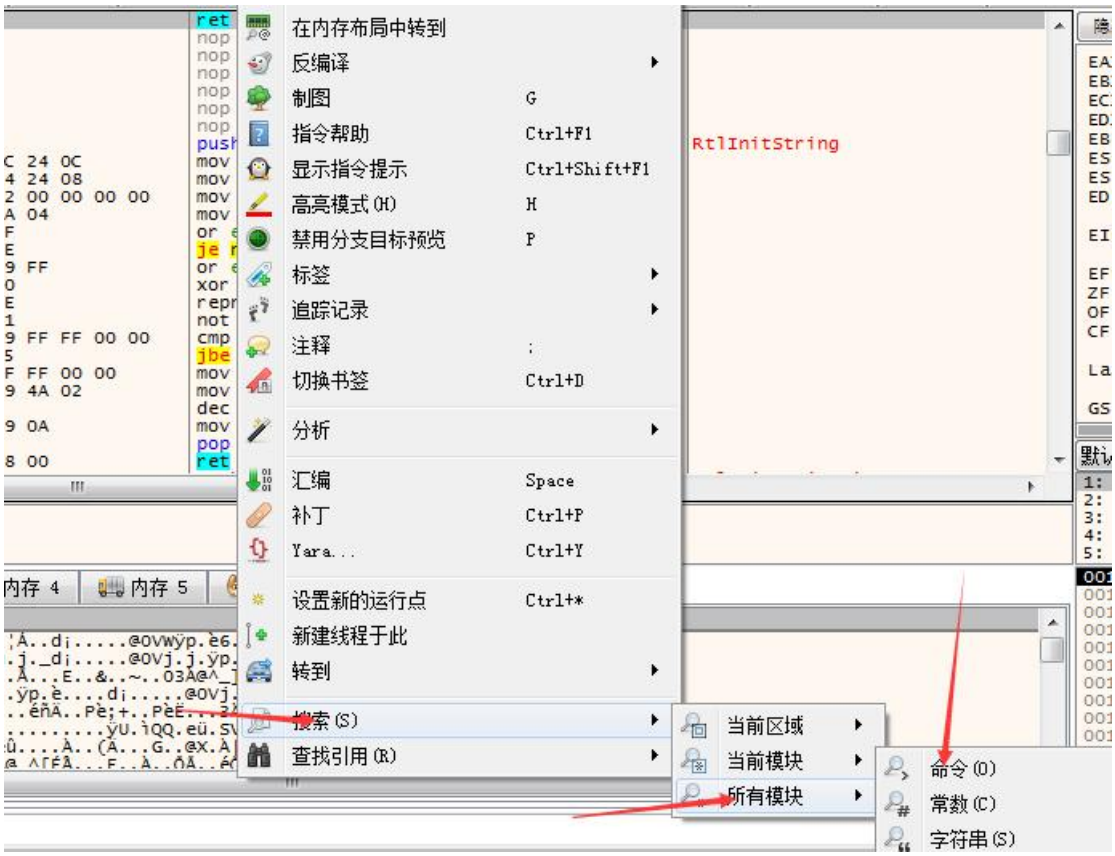


我们发现了作者的跳板地址和后面的 shellcode,由于实验环境不同所以地址偏移也有所不同



回到调试器中观察两个地方 eip 和 esp 这两个值都是可控的了,只需要将 00121004 地址处改

为 jmp esp 的地址,将 00121008 地处开始布置我们的 shellcode,现在进程空间找到一个可用的 jmp esp 的地址,由于格式输出对字符串的处理是 00 阶段所以 地址不可以有 00 出现
为了选择一个合适的跳板地址,这里我们用 x32dbg 找,直接在 ovftool.exe 中运行 exp 程序崩溃选择调试就默认选择了 x32dbg 搜索 call esp



下面是我们查找到的跳板地址我们选择一个不带 00 的还得是可见字符.. 77236928
后面的 shellcode 也必须是字母数字的组合,如何生成这种 shellcode 请参考 alpshellcode 加密